Mixing representation levels: The hybrid approach to automatic text generation

E. Pianta ; L.M. Tovena ;
ITC-IRST ;
via Sommarive
38050 Povo TRENTO - Italy ;
{tovena,pianta}@itc.it

Abstract

Natural language generation systems (NLG) map non-linguistic representations into strings of words through a number of steps using intermediate representations of various levels of abstraction. Template based systems, by contrast, tend to use only one representation level, i.e. fixed strings, which are combined, possibly in a sophisticated way, to generate the final text.
In some circumstances, it may be profitable to combine NLG and template based techniques. The issue of combining generation techniques can be seen in more abstract terms as the issue of mixing levels of representation of different degrees of linguistic abstraction. This paper aims at defining a reference architecture for systems using mixed representations. We argue that mixed representations can be used without abandoning a linguistically grounded approach to language generation.

1 Introduction

Natural language generation systems (NLG) map non-linguistic representations into strings of words through a number of steps using intermediate representations of various levels of abstraction. Template based systems, by contrast, tend to use only one representation level, i.e. fixed strings, which are combined, possibly in a sophisticated way, to generate the final text.

In some circumstances, it may be profitable to combine NLG and template based techniques. The issue of combining generation techniques can be seen in more abstract terms as the issue of mixing levels of representation of different degrees of linguistic abstraction. This paper aims at defining a reference architecture for systems using mixed representations. We argue that using templates does not necessarily means abndoning a linguistically grounded approach to language generation.

The rest of this paper is organised as follows: in section 2 we introduce briefly the NLG and the template based approaches to text generation. Section 3 offers a theoretical framework within which the hybrid approch combining these two strategies can be analysed. We also review a number of existing systems from the point of view of the proposed framework. Finally, section 4 presents a declarative formalism that allows us to represent hybrid objects characterised by different degrees of linguistic abstraction.

2 NLG and templates

The strategy to perform automatic text generation called Natural Language Generation par excellence, or also Deep Generation, is characterised by the fact that it relies on conceptual models of language developed in current linguistic theories. Systems following this strategy are based on concepts such as morpheme, word, sentence, semantic or syntactic representation, communicative intention, etc. Each of these objects pertains to a level of linguistic analysis that has specific rules and specific ways to represent information.

Current architectures of NLG systems are made of an ordered sequence of components consisting of Text Planner, Sentence Planner and Linguistic Realiser (Reiter, 1994; Reiter and Dale, 1997; Paiva, 1998; Cahill and Reape, 1998). Each component expects an input and produces an output pertaining to a certain level of linguistic analysis. More specifically:

- the input of the Text Planner is some information represented in a non-linguistic formalism, for instance concepts of a Knowledge Base. The pieces of information feeding the generation process are usually called *Messages*.
- the output of the Text Planner and input of the Sentence Planner is a *Text Plan*, that is a tree structure where the leaves are Messages and the nodes represent semantic relations between subtrees which will be eventually realized as text spans. In some

case semantic relations correspond to rhetorical relations.

- the output of the Sentence Planner and input of the Linguistic Realiser is a list of *Sentence Representations*. A Sentence Representation includes all the semantic and grammatical information necessary to generate exactly one correct sentence according to the syntactic, morphological and phonological rules of a certain language.

- the ouput of the Linguistic Realiser is a list of strings that constitute the generated text.

The NLG approach is usually compared with the template based one (Reiter, 1995). The basic idea of a textual template is that of a structure consisting of a sequence of fixed strings and gaps filled at processing time with other fixed strings. Let's call it a template of the *static* type. A static template can degenerate in canned text if no gaps are present. Generation systems based solely on static templates are called mail-merge systems. As suggested by Reiter and Dale (1997), these systems can reach the complexity and flexibility of a programming language, and are thus functionally equivalent to NLG systems, at least in principle.

Pros and cons of both approaches have been discussed in the literature, see (Reiter and Mellish, 1993; Reiter, 1995; Reiter and Dale, 1997; Busemann and Horacek, 1998). Let us just mention some of them.

- Pros of NLG: declarativeness, theoretical soundness, modularity, good portability through different domains i.e. reusability, aptness to handle multilinguism.

- Cons of NLG: low time efficiency, architectural complexity, linguistic resources are costly to develop and require specialized knowledge, patches of poorly understood linguistic phenomena, difficulties in integrating linguistic content and lay-out information.

- Pros of static templates: high time efficiency, architectural simplicity, efficient application development, only generic programming skills are required.

- Cons of static templates: no theoretical grounding, procedurality, low portability, multilinguism is awkwardly handled.

In the last few years, templates have been used also within NLG architectures, in order to overcome some of the drawbacks of NLG, first of all time inefficiency and resource developing cost. Reiter and Mellish (1993) introduce pointers to KB individuals among fixed strings and insert canned text as the value of a frame representing the meaning of a sentence. Busemann (1996) mixes templates and syntactic representations. Cancedda et al. (1997) insert templates as leaf nodes of a textual plan produced by classical NLG techniques. These attempts to integrate templates within NLG architectures all bring in *mixed representations* (MR), absent from both pure NLG and static templates.

Our approach to text generation aimes at mixing representation levels in a systematic and principled way. From a practical perspective, this amounts to using precompiled generation knowledge whenever possible, while retaining the possibility of using a full-fledged NLG approach when strictly necessary.

Let us see how the notion of mixed representation compares with the received view about the separation of representation levels. Standard NLG architectures map an input message to an output text passing through a number of intermediate representations, as we have seen. Each representation level is the input and/or output of a separate component coping with specific linguistic phenomena, e.g. communicative intentions, text structure, referring expressions, morphology, etc. In the accepted view, representation levels should be kept carefully separated, on the grounds that separation enhances modularity and reflects different levels of linguistic analysis. In the Mixed Representation approach both these motivations are challenged.

On the one hand, we argue that representation levels can be mixed while preserving the modularity of the linguistic components. On the other hand, we argue that, while the strict separation of representation levels is crucial when taking a competence point of view on language, mixing representations is acceptable in a more performance oriented perspective. In practical terms, we consider it plausible that human speakers produce discourse by mixing dynamic planning with precompiled knowledge about the structure and te relevance of texts, and produce sentences by mixing flexible sentence planning and realization with all kind of (semi)idiomatic expressions, (semi)-fixed descriptions of individuals, precompiled sentence patterns, phrases stored in the short-term memory, etc.

Let us conclude this section by making a terminological point. Instead of the opposition between NLG and templates introduced by Reiter (1995), more recently Busemann and Horacek (1998) have proposed a distinction between in-depth and shallow generation which parallels the distinction between deep and shallow analysis. We think that this proposal goes in the right direction for two reasons. First, identifying natural language generation with deep generation seems misleading and a little arbitrary. Second, as we will see in the rest of the paper, templates are just one among various shallow generation techniques available.

3 The hybrid approach to automatic text generation

In this section we try to characterize the so called hybrid approach to text generation in terms of mixing representation levels. Before doing that, we need to single out a

few more levels of representation.

We start by distinguishing three components which are virtually present in any Linguistic Realiser, that is the Sentence Grammar, the Morphological Synthesizer and the Phonological Adjustment Component. The little attention paid so far to Morphological and Phonological components may be explained by the fact that many of the generation systems described in the literature produce texts in English, a language with relatively little inflectional Morphology and a restricted number of Phonological Adjustment phenomena. In principle, a Sentence Grammar can incorporate the other two components, since it can produce directly complete words and apply phonological adjustment rules as soon as a pair of adjacent words is available. However, we would rather keep the three components apart, in order to broaden the possible range of mixed representations, as discussed below. The distinctions introduced in the Linguistic Realiser implies new representation levels.

- the output of the Sentence Grammar and input of the Morphological Synthesizer is a list of *morphological bundles*, i.e. sets of morphological features that are mapped onto potential words.
- the output of the Morphological Synthesizer and the input of the Phonological Component is a list of *potential words*, which are word forms that can undergo phonological adjustments.
- the output of the Phonological Component is a list of strings.

We must spend a few words also on the task of *formatting*, which is often underestimated in the literature. More and more often generation systems are expected to produce formatted text rather than bare ASCII code. Formatting information is taken in the broad sense of tags for typographical formatting, pointers to images, hypertextual links, annotations for texts feeding a speech synthesizer, etc. In order to be effective, formatting decisions cannot be taken by a component independent from the NLG architecture and subsequent to it. They need to be taken at early stages of the NLG process, see (Reiter et al., 1995). For instance, if you want to emphasise typographically a certain portion of a sentence content, you need to take this decision within the Sentence Planning component, when the relevant semantic information is available. Or, if you want your text to be articulated into sections and paragraphs, you need to take decisions about these aspects during the Text Planning stage. The actual execution of formatting instructions can be left to a component that operates after the Linguistic Realiser, but the formatting decisions themselves need to be taken at the appropriate level of linguistic abstraction.

The finer grained version of the NLG architecture proposed here is made of six components: Text Planner, Sentence Planner, Sentence Grammar, Morphological Synthesizer, Phonological Component, Formatting Realiser. The corresponding representation levels are: (1,Msg)[1] Message, (2,TPlan) Text Plan, (3,SRep) Sentence Representation, (4,MBundle) Morphological Bundle, (5,PWord) Potential Word, (6,Str) String and (7,Frm) Formatting Instructions.[2]

In a hybrid architecture all levels can be mixed. Mixing representation levels can be done either by concatenation or by embedding. *Concatenation* means building a list of objects pertaining to different levels. For instance, one can have a list with the structure [<string>, <potential word>,<sentence representation>,<string>]. Mixing by *embedding* means that an object of a certain level is nested inside a structure of a different level. Only structured objects can be the locus of an embedding. Fixed strings and potential words can be embedded, but one cannot embed into them.

Let us discuss a few cases from the literature in the light of the theoretical framework that we are proposing for hybrid systems. The generation techniques proposed in IDAS (Reiter and Mellish, 1993; Reiter et al., 1995) mix representations levels in two ways. Knowledge base references to entities can be embedded into portions of canned text, which gives a solution of the type [Msg, Str]. They also fill case frame slots with canned text, which corresponds to a type [SRep(Str)].

Busemann (1996) presents TG/2, a surface generator taking as input formulae of the GIL sentence representation formalism. This system is based on production rules employing canned text, templates and syntactic representations. The production rules can contain calls to other rules, lines of Lisp code and canned text. The whole proposal seems to be a solution of the type [Msg, SRep, Str], where messagges are picked up through direct Lisp calls. Then, Busemann and Horacek (1998) do away with the GIL representation interface and replace it with an Intermediate Representation layer that is made up of domain specific conceptual structures. More on this approach in section 4.2.

Geldof and Van de Velde (1997) propose a template based system for generating hypertexts. They use templates made up of canned text interleaved with "abstract terms referring to domain concepts". This type of template corresponds, in their words, to the IDAS' solution of type [Msg, Str] mentioned above. Then, there are templates with hypertext links. Finally, a text schema is used to structure the text. This gives a solution of the type [TPlan([Msg, Str, Frm])].

[1](<level number>,<abbreviation>) are given for easying future reference.

[2]Formatting instructions can be introduced at any level, so we mention level (7,Frm) only as an abstraction.

4 The Hyper Template Planning Language

In order to extend the potentiality of the hybrid approach, Cancedda et al. (1997) developed a specific representation language, Hyper Template Planning Language (HTPL), which allows one to mix together MBundle, PWord, Str and Frm. We call *flexible* templates the kind of structures that can be built by mixing these representation levels. Recall that static templates were defined above as operating on fixed strings.

Then, in (Pianta and Tovena, 1998) the expressive power of HTPL has been extended by adding the possibility to mix also Messages and Sentence Representations. Here is a list of the linguistic representation levels available in the current version of HTPL.

message representation (1,Msg): a formula in some content representation formalism. When specifying a message representation, one should also specify the formalism and the type of message object which is being described. For instance, `msg('IF', attribute, location=pittsburgh)` can be used to refer to an attribute-value pair of the Interchange Format (IF) representation language (Tovena and Pianta, 1999). Message representations are handled by a specialized component during the interpretation process. In the current version of HTPL, messages can by at most proposition level content specifications. Thus, the component that handles them is in fact a Sentence Planner.

phrase representation (3,SRep): an abstract representation of a phrase which can feed a specific tactical generator. For instance, we can specify phrases in terms of grammatical functions such as subject, verb, object, adjuncts, determiner etc., in the spirit of LFG.

```
phrase(lfg,
  [subject=
   [spec=the,num=sing,pred=room]])}.
```

morphological bundle (4,MBundle): a set of morphological features corresponding to a word form. For example, the bundle `morpho([cat=noun, pred=room, num= plur])` can be seen as an abstract representation of the word form `rooms`. When used in HTPL expressions, the values of morphological features can be variables: `morpho([cat=noun, pred=room, num= NUM])`. Morphological variables make it easier to treat agreement phenomena, which are awkward to handle with static templates.

potential word (5,PWord): a word form which can undergo phonological adjustment. We describe a potential word by specifying the lexical category of the word and its base form: `w(noun, albergo)`. Sequences of potential words are mapped onto strings by phonological and orthographic rules: for example in Italian `[w(prep, di), w(article,il), w(noun, albergo)]` becomes `["dell'albergo"]`. The preposition `di` is first combined with the article `il` yielding the compound form `del` (of the). The latter combines with a noun beginning with a vowel yielding a contracted word group which is orthographically represented as "dell'albergo" (of the hotel).

string (6,Str): a sequence of characters inserted in the text without modification, for instance: "hotel reservation".

The representation levels have been listed here following an ordering which is relevant for the HTPL interpreter, see below. However the ordering is also meaningful from a linguistic point of view. A phrase representation is linguistically less abstract than a message representation. A potential word pertains to a less complex constituency level than a phrase representation. A potential word is less abstract than a potential word, etc.

Objects of level 1 through 4 are all parametric, i.e. they can contain variables which are instantiated at processing time. This allows the possibility of sharing information between objects of different levels.

An HTPL expression can include any combination of the above representation levels. Both concatenation and embedding are possible. Here is an example of concatenation:

```
[w(pronoun, 'I'), w(modal, will),
"arrive at", w(article, the),
morpho([cat=noun,
       pred=airport,
       num=sing]),
msg('IF', attribute, time=sunday)]
```

and here is an embedding:

```
phrase(lfg,
  [subject=
     htpl([w(pronoun, 'I')]),
   modality=will
   verb=
     htpl([
       "arrive at",
       w(article, the),
       morpho([cat=noun,
               pred=airport,
               num=sing])]),
   adjuncts=
     htpl([
       msg('IF', attribute,
           time=sunday)])
  ])
```

Both these HTPL expressions correspond to the sentence *I'll arrive at the airport on Sunday*. Of course the first expression can be realized more efficiently than the second, as it doesn't need to be handled by the Sentence Generator. However the second allows more flexibility; under certain conditions, the Sentence Generator could topicalize the adjunct yielding *ON SUNDAY, will I arrive at the airport!*. Also, note that what can be embedded is not simply canned text but any legal HTPL expression. Embedding is explictly marked by enclosing the embedded expression in the scope of the `htpl` operator.

formatting (7,Frm): Typographical formatting phenomena are handled in HTPL by including basic expressions in the scope of one ore more formatting operators such as: `italic`, `bold` etc. Hypertextual links are treated as a special class of format instructions. They are specified by descriptors which refer to linked documents through absolute addresses (file name) or functional expressions, evaluated at run time. Pictures are inserted in text through the same mechanism. Here follow other HTPL objects.

slot specifications: `slot(`<parameters>`)`. The runtime evaluation of a slot specification is expected to yield an HTPL expression.

template definitions: `template(`<template descriptor>, <HTPL expression> `)`. The <template descriptor> can include variables, thus allowing the definition of parametric templates.

control expressions: `if_then, if_then_else, or`. In conditional expressions, an HTPL expression is realized in the generated text only if some constraint is satisfied. Here is an example of a template definition including a conditional expression and a recursive call to other templates:

```
template(controls(ActID),
  if_then_else(
    exist_many_controls(ActID),
    template(item_controls(ActID)),
    template(coord_controls(ActID))))
```

Disjunctive expressions give alternative ways of phrasing something, for example: `or(["taking into account", "considering"])`. When the HTPL interpreter finds a disjunctive expression for the first time, it chooses one of the alternatives randomly; the second time, one of the remaining alternatives is selected, and so on. When all alternatives have been used at least once, the whole set becomes available again.

4.1 The HTPL interpreter

The HTPL interpreter must be able to handle both concatenated and embedded Mixed Representations (MR). As for concatenated MRs, the interpreter scans the list of objects several times. The first time, it calls the component appropriate for all objects of level (1, Msg) that is a Sentence Planner. Then, it passes all the objects of level (2, SRep) to the Sentence Grammar, and so on up to objects of level (6, Str), which are passed to the Formatting Realiser. The latter translates formatting instructions in HTML tags. Note that each component related to a certain level can produce as output MRs, although of a less abstract level.

Handling embedded MRs is more complex, as it depends highly on the working of the single components. For this reason, solving embedded MRs is the responsibility of each component, and won't be further discussed here. The only generic constraint enforced by the HTPL intepreter is that an object of level n should not embed objects of higher abstraction levels.

4.2 Mixed Representations vs Intermediate Representations

In section 3 we already analysed various hybrid approaches to text generation in terms of the MRs framework. In this section we will make some additional comparison between our proposal and that in (Busemann and Horacek, 1998). The two approaches share many practical motivations and adopt a number of similar or equivalent technical solutions. There is one point however that sets well apart the two approaches. Busemann and Horacek (1998) introduce Intermediate Representations, which can be characterized as language independent but domain dependent representations. Notice that the domain dependency holds not only at the level of the concepts of the ontology but also at the level of the syntax and the intepretation of the representation language. In other words, Intermediate Representations are very different both from language dependent grammar representation formalisms such as SPL, and from knowledge representation formalisms based on a general syntax and a general semantic interpretation mechanism. As the authors themselves suggest, the use of these kind of representations seriously undermines the standard NLG architecture as it doesn't acknowledge the text analysis levels on which that architecture is based. We think that the notion of Mixed Representation does not have the same reflexes on the NLG architecture. In our proposal, all analysis levels are kept, indeed some more are made explicit. What is different with respect to the NLG architecture is the possibility to introduce precompiled knowledge at any stage of the generation process. This, from a processing point of view, corresponds to the ability to skip unnecessary intermediate processing stages.

5 Conclusion

In this paper we discussed the hybrid approach to automatic text generation. The concept of mixed linguistic

representation turned out to be a core notion for building a theoretical framework within which to represent different attempts to combine NLG and template based approaches. This conceptual framework led us to propose a more detailed version of the standard NLG architecture and hence new types of mixed representations. These ideas were implemented in HTPL, which has been successfully used in two applicative projects, see (Cancedda et al., 1997; Pianta and Tovena, 1998).

References

Stephan Busemann. Best-first surface realization. In *Eighth International Natural Language Generation Workshop*, pages 101–110, Sussex, 1996.

Stephan Busemann and Helmut Horacek. A flexible shallow approach to text generation. In *Ninth International Natural Language Generation Workshop*, Niagara Falls, 1998.

Lynne Cahill and Mike Reape. Component tasks in applied NLG systems. ms., 1998.

Nicola Cancedda, Gjertrud Kamstrup, Emanuele Pianta, and Ettore Pietrosanti. SAX: Generating hypertext from SADT models. In *Third Workshop on Applications of Natural Language to Information Systems*, Vancouver, 1997.

Sabine Geldof and Walter Van de Velde. An architecture for template based (hyper)text generation. In *Sixth European Natural Language Generation Workshop*, pages 28–37, Germany, 1997.

Daniel Paiva. A survey of applied natural language generation systems. Technical Report 98-03, ITRI, Brighton, 1998.

Emanuele Pianta and Lucia M. Tovena. Generating with flexible templates from C–STAR Interchange Format. Technical Report 9808-04, ITC-IRST, Trento, 1998.

Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Seventh International Workshop on Natural Language Generation*, Boston, 1994.

Ehud Reiter. NLG vs. Templates. In *Fifth European Workshop on Natural Language Generation*, Leiden, 1995.

Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3:57–87, 1997.

Ehud Reiter and Chris Mellish. Optimising the cost and benefits of natural language generation. In *IJCAI-1993*, pages 1164–1169, San Mateo Cal., 1993.

Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of technical documentation. *Applied artificial intelligence*, 9:259–287, 1995.

Lucia M. Tovena and Emanuele Pianta. Generating felicitous sentences from underspecified semantic representations. In *Proceedings of the 3rd International Workshop on Computational Semantics*, pages 410–412, Tilburg, 1999.